**Attorney Docket No.**
**OI7030492001**

ORACLE CONFIDENTIAL

UNITED STATES PATENT APPLICATION

FOR

# METHOD AND MECHANISM FOR PERFORMING A ROLLING UPGRADE OF DISTRIBUTED COMPUTER SOFTWARE


INVENTORS:

**ALOK KUMAR SRIVASTAVA**
**RAJIV JAYARAMAN**

PREPARED BY:

**PETER C. MEI**
**BINGHAM MCCUTCHEN LLP**
**THREE EMBARCADERO CENTER, SUITE 1800**
**SAN FRANCISCO, CA 94111-4067**

ASSIGNEE: ORACLE INTERNATIONAL CORPORATION
500 ORACLE PARKWAY
REDWOOD SHORES, CA 94065


EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number EV348159766US

Date of Deposit March 17, 2004
I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to MAIL STOP PATENT APPLICATION, Commissioner for Patents, Alexandria, VA 22313-1450.

Linda Major
(Typed or printed name of person mailing paper or fee)

(Signature of person mailing paper or fee)

# METHOD AND MECHANISM FOR PERFORMING A ROLLING UPGRADE OF DISTRIBUTED COMPUTER SOFTWARE

## BACKGROUND AND SUMMARY

**[0001]** The invention relates to a method and mechanism for performing rolling software upgrades to a distributed computing system.

**[0002]** Over time, many types of software applications will undergo some sort of change. These changes may occur for a variety of reasons. For example, given the complexity of modern software applications, it is well known that most software contains coding errors or "bugs" that will need correction. One reason to upgrade from an earlier version to a later version of a software application is to correct errors or bugs that may exist in the earlier version. Another reason for changing a software application is to introduce improvements to the operation or functionality of the software application.

**[0003]** A "rolling upgrade" refers to the process of performing software upgrades to a live existing software installation in a distributed environment in which the individual instances, nodes, or entities of the distributed system (referred to herein as "members") are upgraded in a staggered manner. This form of upgrade ensures availability of the application during software upgrades, and thus minimizes or eliminates planned downtime while contributing to high availability goals. As used herein, the term member may

encompass either a single instance/node/entity or a collection of such instances/nodes/entities.

**[0004]** At each member, there are numerous ways to upgrade a software application from an earlier version to a later version. A common approach is for a software developer to create patches and patch sets that are applied to a copy of the software binary or executable. Another common approach is to create a new object having the same location reference. Tools are often provided to perform the software upgrades or installations.

**[0005]** Performing an upgrade or change to an existing software application typically requires a shutdown of either/both the member or software enterprise. For example the upgrade can be performed by shutting down the software, implementing the upgrade, and then bringing the member back up so that availability is restored.

**[0006]** With modern software, it can be anticipated that software developers will provide upgrades and changes on an ongoing basis. In fact, many IT ("information technology") departments will periodically schedule planned events to perform upgrades to their software installations. These events could result in significant planned downtimes. It is desirable to limit the effects of these downtimes as much as possible since they could affect the availability of mission critical systems, potentially resulting in productivity and financial losses for organizations.

**[0007]** If the system being upgraded is a distributed system having multiple independent members where the software is located in the members' local directories, then in one approach, the upgrade can be performed individually at each member so that other members

do not suffer downtime while the affected member is being upgraded. However, problems arise with this approach if it is implemented in networked and shared filesystem environments in which multiple members operate with the same shared software installations. Some examples of this type of configuration are when multiple members access the same software installation at a shared filesystem using the NFS (network file system) mechanism or in the Cluster File System such as the Oracle Cluster File System (OCFS) available from Oracle Corporation of Redwood Shores, California. With this type of architecture, since the application files are shared, performing a rolling upgrade could result in all members being shutdown during the upgrade process, resulting in total unavailability for the systems during the downtime.

**[0008]** For operating system (OS) upgrades, one approach for handling this is provided in the Tru64/TruCluster system which offers OS level support to perform rolling upgrades on their Cluster File System. The TruCluster model uses tagged files and kernel parameters to support multi-versioning and version switching. However, this approach may result in inefficient performance involving 2n-1 reboots to the networked members when performing the OS upgrades where n is the number of members being upgraded.

**[0009]** Therefore, to address these and other problems, what is described herein is an improved method and mechanism for performing rolling upgrades, e.g., to shared software installations in a distributed environment. The present approach eliminates or minimizes extraneous downtime when performing a rolling upgrade, thereby improving performance and availability for users of the shared software installation. In one embodiment, a rolling

upgrade is performed by defining a private symbolic link for each member that is upgraded to reference the upgraded version of the shared software installation. This approach can be performed upon any computing system, whether single node (e.g., a multi-instance application on a single computer) or a multi-node system (e.g., a cluster or network of stations).

**[0010]** Further details of aspects, objects, and advantages of the invention are described below in the detailed description, drawings, and claims.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0011]** The accompanying drawings are included to provide a further understanding of the invention and, together with the Detailed Description, serve to explain the principles of the invention. The same or similar elements in the figures may be referenced using the same reference numbers.

**[0012]** Fig. 1 shows an example of a system in which multiple members access the same shared files.

**[0013]** Fig. 2A generally illustrates a private symbolic link according to an embodiment of the invention.

**[0014]** Fig. 2B generally illustrates a member private symbolic link according to an embodiment of the invention.

**[0015]** Figs. 3A and 3B show flowcharts of processes for performing rolling upgrades to a software installation according to embodiments of the invention.

**[0016]** Figs. 4A-G and 5A-I illustrate processes for performing rolling upgrades.

**[0017]** Fig. 6 is a diagram of a computer system with which the present invention can be implemented.

## DETAILED DESCRIPTION

**[0018]** The present invention provides a method and mechanism for performing a rolling upgrade to shared software installations in a distributed environment, e.g. for a networked, shared filesystems. The present approach eliminates or minimizes extraneous downtime when performing a rolling upgrade, thereby improving performance and availability for users of the shared software installation.

**[0019]** Fig. 1 shows an example of a system 100 in which multiple networked members 1, 2, 3 access shared files 106 in a shared file system. Assume that each of the members 1, 2, and 3 are servers running an enterprise software that support a large number of clients 102 in the system 100. The installed software application running on each of the members 1, 2, and 3 are executed from the shared files 106 in the shared file system rather than from application executables in their private file systems.

**[0020]** In this type of architecture, performing a rolling upgrade could cause significant downtime in the system 100. This is because one has to stop all the instances of the application running the distributed members 1, 2, 3 to change or upgrade the installed software in the shared files 106. In other words, none of the clients 102 will be able to perform work in the system while an upgrade is being performed in this approach, since all members that require access to shared files 106 would be down.

**[0021]** Embodiments of the present invention provide an improved method and mechanism for performing rolling upgrades that do not require all members using the shared

file system to be brought down when upgrading or changing installed software applications. In one embodiment, the rolling upgrade is performed using private symbolic links (PSL).

**[0022]** At this point, it is helpful to generally describe a symbolic link. A symbolic link is a logical reference from a first file/pathname to a second file/pathname, which is most commonly used in UNIX-based systems. Most operations (e.g., read and execute operations) that are called to operate upon the symbolic link are instead redirected to operate upon the filename referenced by the symbolic link.

**[0023]** In a present embodiment, a private symbolic link mechanism is used to perform a rolling upgrade. Unlike a standard symbolic link, a PSL does not provide a universally visible symbolic link between two file/pathnames. Instead, the PSL makes the link between two filenames visible only to members specified or authorized for the PSL. The present disclosure defines the semantics of a PSL that is employed in one embodiment of the invention for performing rolling upgrades.

**[0024]** Fig. 2A illustrates this aspect of a PSL. In Fig. 2A, a networked storage device 200 includes a shared file system. Three members 206, 208, and 210 are shown which access the shared file system. Assume that the pathname of the shared file that is accessed by all three members is "/binaries/version". Three different versions of the file associated with the pathname "/binaries/version" have been stored in networked storage device 200. A first version 202a actually has the pathname "/binaries/version1", a second version 202b has the pathname "/binaries/version2", and a third version 202c has the pathname "/binaries/version3".

**[0025]** It is desired to establish a configuration of the system such that different entities in the system will symbolically link from the same pathname/filename to different files in the shared file system. This is accomplished in the present embodiment by establishing one or more private symbolic links in which a link criteria or link configuration will determine which file will be referenced by the symbolic link for particular entities in the system. As used herein, such entities include any object, member, application or individual which is capable of using or accessing a shared file. For the purposes of illustration, the term "member" will be used instead of "entities" in the rest of this description.

**[0026]** An example structure 220 is shown in Fig. 2A which identifies some example link configuration data for a private symbolic link in this example. In the present example, the structure 220 includes "link criteria" information which contains the matching rule or criteria that identifies which entities or categories of entities corresponds to the PSL definition. This means that the PSL will only be applied to an entity if that entity is a member of the group defined by the link criteria information for the PSL. More than one entity or member may be a member of this group, and therefore associated with the same PSL. The "link from" information identifies the symbolic link name and/or the name for which linking is desired. The "link to" information identifies the target filename or pathname for the private symbolic link. It is noted that structure 220 is merely illustrative, and not meant to be limiting, since any suitable link configuration approach that may be employed, including modification of OS/directory files to establish or define a private

symbolic link. As can be appreciated by those skilled in the art, it is clear that any number of approaches may be used to implement configuration data for a private symbolic link.

**[0027]** In structure 220, a first entry 222 has been configured to include a link criteria definition that applies to all members having a given software application at version 1. This means that the PSL associated with entry 222 only applies to a member only if the member has the stated software application at version 1. If the member is running the software application at any other version number, then the PSL defined by entry 222 does not apply to that member. Entry 222 is configured such that it "link from" a filename/pathname for "/binaries/version". The "link to" information for entry 222 has been defined as "/binaries/version1". This means that any member that is a member of the group defined by the link criteria information for entry 222 will have a private symbolic link from the "/binaries/version" file/pathname to the "/binaries/version1" file/pathname.

**[0028]** Similarly, it can be seen that entry 224 has link criteria information that defines a PSL which applies to members running the software application at version 2. Entry 224 is configured such that it "link from" a filename/pathname for "/binaries/version". The "link to" information for entry 224 has been defined as "/binaries/version2". This means that any member that is a member of the group defined by the link criteria information for entry 224 will have a private symbolic link from the "/binaries/version" file/pathname to the "/binaries/version2" file/pathname. Entry 226 has link criteria information that defines a PSL which applies to members running the software application at version 3. Entry 226 is configured such that it "link from" a filename/pathname for "/binaries/version". The "link

to" information for entry 226 has been defined as "/binaries/version3". This means that any member that is a member of the group defined by the link criteria information for entry 226 will have a private symbolic link from the "/binaries/version" file/pathname to the "/binaries/version3" file/pathname.

**[0029]** In the example of Fig. 2A, assume that member 206 is running the software application at version 1, member 208 is running the software application at version 2, and member 210 is running the software application at version 3. Since member 206 is a member of the group for the PSL defined by entry 222, when member 206 seeks to access the shared file "/binaries/version", a private symbolic link 230 will instead cause member 206 to access the file 202a corresponding to the file/pathname "/binaries/version1". Since member 208 is a member of the group for the PSL defined by entry 224, when member 208 seeks to access the shared file "/binaries/version", a private symbolic link 232 will instead cause member 208 to access the file 202b corresponding to the file/pathname "/binaries/version2". Similarly, member 210 is a member of the group for the PSL defined by entry 226, and therefore, when member 210 seeks to access the shared file "/binaries/version", a private symbolic link 234 will instead cause member 210 to access the file 202c corresponding to the file/pathname "/binaries/version3".

**[0030]** One specific type/variant of a PSL that can be created is a member private symbolic link (MPSL). The MPSL creates a symbolic link that only applies to the member defined for the MPSL. Therefore, the member of the group defined for a MPSL may be limited to a single identified member. Fig. 2B shows an example of a configuration

structure 250 that may be used to define a set of MPSLs. Structure 250 includes "member" definition information which contains the identity of the particular member corresponding to a MPSL definition. This means that the MPSL will only be applied to a member if that member is associated or defined for the MPSL. As before, the "link from" information identifies the symbolic link name and/or the name for which linking is desired. The "link to" information identifies the target filename or pathname for the private symbolic link.

**[0031]** Entry 252 in structure 250 has the name "member 1" defined in the member column. This means that the PSL associated with entry 252 only applies to the member identified as "member 1". Entry 252 is configured such that it "link from" a filename/pathname for "/binaries/version". The "link to" information for entry 252 has been defined as "/binaries/version1". This means that member 1 will have a private symbolic link from the "/binaries/version" file/pathname to the "/binaries/version1" file/pathname. Similarly, it can be seen that entry 254 has been defined to only apply to the member named "member 2". Entry 254 is configured such that it "link from" a filename/pathname for "/binaries/version". The "link to" information for entry 254 has been defined as "/binaries/version2". This means that member 2 will have a private symbolic link from the "/binaries/version" file/pathname to the "/binaries/version2" file/pathname. Entry 256 has been defined to only apply to the member named "member 3". Entry 256 is configured such that it "link from" a filename/pathname for "/binaries/version". The "link to" information for entry 256 has been defined as "/binaries/version3". This means that member 3 will have

a private symbolic link from the "/binaries/version" file/pathname to the "/binaries/version3" file/pathname.

**[0032]** In the examples of Fig. 2A and 2B, although "/binaries/version" resolves into the private copies "/binaries/version[1,2,3]", these copies are stored on the shared storage but are not accessible except by members that meet the link criteria.

**[0033]** The private symbolic link can be used to implement a process for performing rolling upgrades to software. Fig. 3A shows a flowchart of a process for implementing a rolling upgrade according to an embodiment of the invention. At 332, a shared copy of the new version of the software is created in the shared file system without removing the old version of the software. As noted previously, there are numerous approaches that can be taken to create the new version of the software. One approach is to copy the old version of the software to another part of the shared file system, and to apply patches or a patch set associated with the new version to the software copy. Another approach is to move or create an entirely new version of the software without copying from the old version. In either case, the new version of the software (or specific files associated with the new version) now co-exists in the shared file system with the old version of the software, but having a different pathname or filename from the old version.

**[0034]** In the approach of Fig. 3A, only a single copy of the new version of the software is used in the shared file system. During the rolling upgrade, each member that is to be upgraded will access this single copy of the new version of the software.

**[0035]** At 334, links are configured such that members that are to be upgraded to the new version of the software will point to the newly created copy of the new version of the software. For a rolling upgrade, this process of pointing the members will occur in a staggered manner such that availability to at least one or more members is retained throughout the upgrade process. A determination is made whether there are any additional members to process (335). If so, then action 334 is repeated for each additional member to create a link to the new version. At 336, the links are deleted after the members point to the new version of the software.

**[0036]** In an embodiment, the MPSL mechanism is used to providing this linking capability. With this approach, a member is identified for which the rolling upgrade is desired. During the upgrade, the member and/or its running software application is brought down. It is noted that in this approach, only the specific member presently identified for the rolling upgrade is brought down. All other members still operating against the old version of the software can continue running without interruption. After the member has been brought down, a private symbolic link is created for the identified member. The private symbolic link creates a symbolic link between the file/pathname of the old version and the file/pathname of the new version of the software. The private symbolic link only applies to the members of the group associated with the MPSL. In one embodiment, the private symbolic link is a member private symbolic link, and the only member of the group associated with the MPSL is the identified member. After linking occurs, the member is brought back up. Because of the MPSL, the software application will be brought up

automatically referencing the application files from the new version of the software. A determination is made whether additional members need to be upgraded during the rolling upgrade. If so, the process is repeated until all members that need to be upgraded have been upgraded.

**[0037]** Once all relevant members have been upgraded, a determination is made whether to remove the old version of the software (338). There are many reasons not to remove the old software. For example, it is possible that there is still a reason to continue using the old software version on some of the members in the network. If, however, the decision has been that the old software version is no longer needed, then the old software version is removed (340).

**[0038]** In the embodiment of Fig. 3A, all members that upgrade will have a private symbolic link that references the same copy of the new version of the software. In an alternate embodiment, multiple copies of the files for new version(s) of the software may concurrently exist. In this situation, the different members may have different sets of private symbolic links that point to different sets of files. This is useful, for example, if the configuration files differ between members, even for the same version of the software. Fig. 3B shows a process flow for this alternate approach.

**[0039]** At 322, a member creates a private copy of a new version of the software. The private copy is created in the shared file system without removing the old version of the software. Any appropriate approach can be taken to create the new private copy. The newly created copy of the new version of the software co-exists in the shared file system with the

old version of the software, but having a different pathname or filename from the old version.

**[0040]** In the approach of Fig. 3B, multiple private copies of the new version(s) of the software can be created in the shared file system. In an embodiment, each member can be associated with its own private copy. Alternatively, even though multiple private copies are created, each private copy is capable of being associated with multiple members. During the rolling upgrade, each member that is to be upgraded will access the appropriate private copy relevant for the type of upgrade desired.

**[0041]** At 324, a link is configured such that the member that is to be upgraded will point to the newly created private copy of the new version of the software. For a rolling upgrade, this process of pointing the members will occur in a staggered manner such that availability to at least one or more members is retained throughout the upgrade process. The MPSL mechanism can be used to providing this linking capability. After linking occurs, the member is brought back up. Because of the MPSL, the software application will be brought up automatically referencing the application files from the new version of the software.

**[0042]** A determination is made whether there are any additional members to upgrade (323). If so, then another private copy of the new version is created (322) and a link is created to the new private copy (324). The process is repeated until all members that need to be upgraded have been upgraded. At 326, the link(s) are deleted after the member(s) point to the new version of the software. At a later point, the members can be configured to

access the same copy of the new version of the software, rather than their own private copies.

**[0043]** Once all relevant members have been upgraded, a determination is made whether to remove the old version of the software (328). If the decision has been that the old software version is no longer needed, then the old software version is removed (330).

**[0044]** To illustrate the embodiment of the invention of Fig. 3A when performing a rolling upgrade, reference is now made to the system diagram of Fig. 4A. In this figure, two members (member 1 and member 2) are shown which run a software application using shared application files 406a from a shared file system. The present version of the software application is version 1. Consider if it is desired to perform a rolling upgrade from version 1 to version 2.

**[0045]** Moving to Fig. 4B, the first action is to create the files 406b corresponding to the new version of the software. In one embodiment, this accomplished by copying the old version 406a of the files to a new location on the shared file system within storage device 404. The patches associated with the new version of the software is applied to the copied software to form the new version 406b of the software application.

**[0046]** At this point, the first member to upgrade is identified. Assume that the first member to upgrade is member 1. Referring to Fig. 4C, member 1 is brought down. Several different approaches can be taken to bring down member 1. One approach is merely to shut down the software application. This approach may be particularly appropriate for higher-level software applications. Another approach is to entirely shut down the hardware at the

node corresponding to member 1. This approach may be more appropriate for OS software. Other and additional approaches can be taken depending upon the specific requirements of the software application being upgraded.

**[0047]** A private symbolic link is created to associate member 1 with the new version 406b of the software. This is shown in Fig. 4C by adding entry 410 to PSL structure 408. In particular, member 1 is identified as the entity associated with the PSL corresponding to entry 410. The "link to" information for this entry 410 identifies the filename/pathname of the application file(s) for the new version 406b of the software. The "link from" information identifies the original file/pathname that the application is configured to access.

**[0048]** Referring to Fig. 4D, the next action is to bring up member 1. Because of the new PSL corresponding to entry 410, the application software will automatically start up based upon the application files associated with the new version 406b of the software.

**[0049]** At this point, it can be seen that members running both the old and new versions of the same software application are simultaneously running the system. In addition, it is noted that when performing the rolling upgrade, the process can be configured such that downtime only occurs for a single member at a time. However, multiple members can still be simultaneously shut down for upgrades within the scope of the invention. The advantage of this approach over the prior approaches is that less than all of the members need be brought down when performing a rolling upgrade for shared application files.

**[0050]** The next action is to upgrade member 2, which is still operating using the old version 406a of the software. Referring to Fig. 4E, member 2 is brought down. Once

member 2 has been brought down, a private symbolic link is created to associate member 2 with the new version 406b of the software. This is shown in Fig. 4E by adding entry 412 to PSL structure 408. In particular, member 2 is identified as the entity associated with the PSL corresponding to entry 412. The "link to" information for this entry 412 identifies the filename/pathname of the application file(s) for the new version 406b of the software. The "link from" information identifies the original file/pathname that the application is configured to access.

**[0051]** Referring to Fig. 4F, the next action is to bring up member 2. Because of the new PSL corresponding to entry 412, the application software will automatically start up based upon the application files associated with the new version 406b of the software.

**[0052]** Since all members have now been upgraded, it is now possible to remove the old version 406a of the software and to remove the links. Turning to Fig. 4G, this figure shows the old version of the software 406a and the link structures being removed. At this point, the system is configured such that all future requests to access the software at the specified reference location will access the new version 406b of the software. The rolling upgrade process now ends.

**[0053]** The process of Fig. 3B can be similarly implemented, differing in that multiple private copies of new version(s) of the software can be created. Thus, in this alternate approach, both member 1 and member 2 can be configured to point to different copies of the upgraded software. To illustrate the embodiment of the invention of Fig. 3B, reference is now made to the system diagram of Fig. 5A. In this figure, two members (member 1 and

member 2) are shown which run a software application using shared application files 506a from a shared file system. The present version of the software application is version 1. Consider if it is desired to perform a rolling upgrade from version 1 to version 2.

**[0054]** Moving to Fig. 5B, the first action is to create the files 506b corresponding to a first private copy of the new version of the software. In one embodiment, this accomplished by copying the old version 506a of the files to a new location on the shared file system within storage device 504. The patches associated with the new version of the software is applied to the copied software to form the new version 506b of the software application.

**[0055]** At this point, the first member to upgrade is identified. Assume that the first member to upgrade is member 1. Referring to Fig. 5C, member 1 is brought down. Several different approaches can be taken to bring down member 1. One approach is merely to shut down the software application. This approach may be particularly appropriate for higher-level software applications. Another approach is to entirely shut down the hardware at the node corresponding to member 1. This approach may be more appropriate for OS software. Other and additional approaches can be taken depending upon the specific requirements of the software application being upgraded.

**[0056]** A private symbolic link is created to associate member 1 with the new version 506b of the software. This is shown in Fig. 5C by adding entry 510 to PSL structure 508. In particular, member 1 is identified as the entity associated with the PSL corresponding to entry 510. The "link to" information for this entry 510 identifies the filename/pathname of

the application file(s) for the new version 506b of the software. The "link from" information identifies the original file/pathname that the application is configured to access.

**[0057]** Referring to Fig. 5D, the next action is to bring up member 1. Because of the new PSL corresponding to entry 510, the application software will automatically start up based upon the application files associated with the new version 506b of the software.

**[0058]** The next action is to upgrade member 2, which is still operating using the old version 506a of the software. Another private copy 506c of the new version of the software is created. Referring to Fig. 5E, member 2 is brought down. Once member 2 has been brought down, a private symbolic link is created to associate member 2 with the new private copy 506c of the new version of the software. This is shown in Fig. 5E by adding entry 512 to PSL structure 508. In particular, member 2 is identified as the entity associated with the PSL corresponding to entry 512. The "link to" information for this entry 512 identifies the filename/pathname of the application file(s) for the new private copy 506c of the software. The "link from" information identifies the original file/pathname that the application is configured to access.

**[0059]** Referring to Fig. 5F, the next action is to bring up member 2. Because of the new PSL corresponding to entry 512, the application software will automatically start up based upon the application files associated with the new version 506c of the software.

**[0060]** At this point, the system is configured such that all future requests to access the software at the specified reference location will access either private copy 506b or 506c of the new version of the software. Since all members have now been upgraded, it is now

possible to upgrade the old version 506a of the software to the new version. Turning to Fig. 5G, this figure shows the system after upgrading the shared copy the software 506a.

**[0061]** Referring to Fig. 5H, the links to the private copies 506b and 506c of the new version can be removed, allowing the members 1 and 2 to point to the shared copy 506a of the upgraded version of the software. At this point, the system is configured such that all future requests to access the software at the specified reference location will access the shared copy 506a of the new version of the software.

**[0062]** Since the members 1 and 2 no longer link to the private copies 506b and 506c of the software, these private copies can be removed. Fig. 5I shows the deletion of the private copies 506b and 506c. The rolling upgrade process now ends.

**[0063]** Multiple versions of the software application can co-exist in the distributed system through the rolling upgrade process. Since member private symbolic links are used, each member may point to different versions of the software. Therefore, if it is desired to leave individual members at different versions of the software, this can be accomplished without interfering with the ability of other members to point to other versions of the software.

**[0064]** The present approach can be applied not only to distributed systems in which the members are on different network nodes, but can also be applied to perform rolling upgrades in which multiple entities occupy the same network node. This type of situation may occur, for example, for a multi-instance application that resided on a single node. The present approach can be used to ensure a high level of availability for the multiple instances on the

single network node during a rolling upgrade be creating private symbolic links for the individual instances, as described with respect to Figs. 3A or 3B.

**[0065]** Therefore, what has been described is an improved process for performing a rolling upgrade to software. It is noted, however, the principles described herein are applicable to a wide variety of different applications in which file branching is desired. For example, the present application may be applicable to situations in which the shared files are data files, rather than shared application executables. This occurs, for example, with stored web pages at a central server. Consider if a first member operate a first type of web browser and a second member operates a second type of web browser, in which it is desirable to maintain a different version of the same web page for each type of browser. In this situation, a private symbolic link could be used to automatically allow each member to correctly reference the specific web page corresponding to its browser type, even though both members may actually attempt to access the same file/pathname or web page name.

**[0066]** When the software to access the shared data files are upgraded, the above process can be used to provide seamless changes to the specific data files accessed by the upgraded applications. Again using the web browser application as an example, consider a browser application that is changed from an earlier version to a later version on a network member. Assume that it is desirable to maintain a different version of the same web page for each version of browser. When the member upgrades to the new version of the browser, a private symbolic link can be defined to allow the upgraded to member to automatically access the correct web page after the upgrade.

## SYSTEM ARCHITECTURE OVERVIEW

**[0067]** The execution of the sequences of instructions required to practice the invention may be performed in embodiments of the invention by a computer system 1400 as shown in Fig. 6. As used herein, the term computer system 1400 is broadly used to describe any computing device that can store and independently run one or more programs. In an embodiment of the invention, execution of the sequences of instructions required to practice the invention is performed by a single computer system 1400. According to other embodiments of the invention, two or more computer systems 1400 coupled by a communication link 1415 may perform the sequence of instructions required to practice the invention in coordination with one another. In order to avoid needlessly obscuring the invention, a description of only one computer system 1400 will be presented below; however, it should be understood that any number of computer systems 1400 may be employed to practice the invention.

**[0068]** Each computer system 1400 may include a communication interface 1414 coupled to the bus 1406. The communication interface 1414 provides two-way communication between computer systems 1400. The communication interface 1414 of a respective computer system 1400 transmits and receives signals, e.g., electrical, electromagnetic or optical signals, that include data streams representing various types of information, e.g., instructions, messages and data. A communication link 1415 links one computer system 1400 with another computer system 1400. A computer system 1400 may

transmit and receive messages, data, and instructions, including program, i.e., application, code, through its respective communication link 1415 and communication interface 1414. Received program code may be executed by the respective processor(s) 1407 as it is received, and/or stored in the storage device 1410, or other associated non-volatile media, for later execution.

**[0069]** In an embodiment, the computer system 1400 operates in conjunction with a data storage system 1431, e.g., a data storage system 1431 that contains a database 1432 that is readily accessible by the computer system 1400. The computer system 1400 communicates with the data storage system 1431 through a data interface 1433. A data interface 1433, which is coupled to the bus 1406, transmits and receives signals, e.g., electrical, electromagnetic or optical signals, that include data streams representing various types of signal information, e.g., instructions, messages and data. In embodiments of the invention, the functions of the data interface 1433 may be performed by the communication interface 1414.

**[0070]** Computer system 1400 includes a bus 1406 or other communication mechanism for communicating instructions, messages and data, collectively, information, and one or more processors 1407 coupled with the bus 1406 for processing information. Computer system 1400 also includes a main memory 1408, such as a random access memory (RAM) or other dynamic storage device, coupled to the bus 1406 for storing dynamic data and instructions to be executed by the processor(s) 1407. The main memory 1408 also may be used for storing temporary data, i.e., variables, or other intermediate information during

execution of instructions by the processor(s) 1407. The computer system 1400 may further include a read only memory (ROM) 1409 or other static storage device coupled to the bus 1406 for storing static data and instructions for the processor(s) 1407. A storage device 1410, such as a magnetic disk or optical disk, may also be provided and coupled to the bus 1406 for storing data and instructions for the processor(s) 1407. A computer system 1400 may be coupled via the bus 1406 to a display device 1411, such as, but not limited to, a cathode ray tube (CRT), for displaying information to a user. An input device 1412, e.g., alphanumeric and other keys, is coupled to the bus 1406 for communicating information and command selections to the processor(s) 1407.

**[0071]** According to one embodiment of the invention, an individual computer system 1400 performs specific operations by their respective processor(s) 1407 executing one or more sequences of one or more instructions contained in the main memory 1408. Such instructions may be read into the main memory 1408 from another computer-usable medium, such as the ROM 1409 or the storage device 1410. Execution of the sequences of instructions contained in the main memory 1408 causes the processor(s) 1407 to perform the processes described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and/or software.

**[0072]** The term "computer-usable medium" or "computer-readable medium" as used herein, refers to any medium that provides information or is usable by the processor(s) 1407.

Such a medium may take many forms, including, but not limited to, non-volatile, volatile and transmission media. Non-volatile media, i.e., media that can retain information in the absence of power, includes the ROM 1409, CD ROM, magnetic tape, and magnetic discs. Volatile media, i.e., media that can not retain information in the absence of power, includes the main memory 1408. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise the bus 1406. Transmission media can also take the form of carrier waves; i.e., electromagnetic waves that can be modulated, as in frequency, amplitude or phase, to transmit information signals. Additionally, transmission media can take the form of acoustic or light waves, such as those generated during radio wave and infrared data communications.

**[0073]** In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. For example, the reader is to understand that the specific ordering and combination of process actions shown in the process flow diagrams described herein is merely illustrative, and the invention can be performed using different or additional process actions, or a different combination or ordering of process actions. The specification and drawings are, accordingly, to be regarded in an illustrative rather than restrictive sense.